1988

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

OPTIMIZATION OF LARGE MATRIX CALCULATIONS FOR EXECUTION ON THE CRAY X-MP VECTOR SUPERCOMPUTER

Prepared by:	Dr. William A. Hornfeck
Academic Rank:	Professor
University and Department:	Mississippi State University Electrical Engineering Department
NASA/MSFC	
Office:	Information Systems
Division:	Systems Development and Implementation
Branch:	Engineering Systems
MSFC Colleague:	Mr. Bobby C. Hodges
Date:	July 29, 1988
Contract No.:	NGT 01-002-099 The University of Alabama

OPTIMIZATION OF LARGE MATRIX CALCULATIONS FOR EXECUTION ON THE CRAY X-MP VECTOR SUPERCOMPUTER

by

William A. Hornfeck
Professor of Electrical Engineering
Mississippi State University

ABSTRACT

A considerable volume of large computational computer codes have been developed for NASA over the past twenty-five years. This code represents algorithms developed for machines of an earlier generation. With the emergence of the vector supercomputer as a viable, commercially available machine, an opportunity exists to evaluate optimization strategies to improve the efficiency of existing software. This result is primarily due to architectural differences in the latest generation of "large-scale" machines and the earlier, mostly uniprocessor, machines. This report describes a software package being used by NASA to perform computations on large matrices, and describes a strategy for conversion to the Cray X-MP vector supercomputer.

ACKNOWLEDGEMENTS

There were quite a large number of persons at MSFC who proved to be very capable and quite eager to provide assistance and motivation during the ten-week course of this study. Certainly my colleague in this effort, Mr. Bobby C. Hodges, has my sincere thanks. Thanks also to his fellow NASA Information System Office personnel John C. Lynn, Director, Sheila Fogle, Shirley Thompson, Carla Krivutza, and Joe Pollock. From NASA's Structures and Dynamics Laboratory, I want to thank John Admire and Dave McGhee who loaned their expertise and computer codes.

I am grateful to a cross-section of NASA Contractor personnel, all of whom were extremely cooperative: Dale Robertson and Deborah Hagar of Grumman Data Systems, Larry Hoelzeman of Cray Research, Karin Offik of New Technology Inc., and Les Wade of Boeing Computer Support Services.

Ernestine Cothran of the MSFC Director's Executive Staff and Dr. Mike Freeman of the University of Alabama performed an outstanding service in coordinating and administering all aspects of the Summer Faculty Program.

INTRODUCTION

The FORMA (Fortran Matrix Analysis) software package was developed by Martin-Marietta approximately twenty years ago. This package has been adapted by NASA for use by the Flight Dynamics Laboratory at MSFC in solving large structures response equations:

$$-W^2 M + S\phi = 0$$

Where ϕ = Mode

M = Mass Matrix

S = Stiffness Matrix

W = System Eigenvalues

$$L(T) = A\frac{d^2X}{dT^2} + B\frac{dX}{dT} + CX + DF(T) + E$$

Where L = Load Matrix

A,...,E = Constant

T = Time

X = Position

F = Forcing Function

and Maximum Dimensions = (12,000 X 12,000)

Typical Matrix Dimensions = (500 X 500)

Atypical Matrix Dimensions = (5,000 X 5,000)

Original FORMA codes were adapted for execution on the MSFC UNIVAC 1108 Multiprocessor. These codes have been "ported" to a next-generation UNIVAC machine, then the IBM 3084, and now the Cray X-MP. Conversions were accomplished in a minimum of time, but without attention to optimization strategies regarding the host machines. The Cray is particularly sensitive to vector constructs within programs.

OBJECTIVES

Develop and adapt specialized mathematical/engineering techniques or methodologies to the solution of scientific/engineering problems utilizing supercomputer technology. Mathematical analyses and modeling of large computerized programs will be performed and recommendations for optimizing the solutions will be formulated. Oral and written reports will be presented/developed on research activities and results.

THE COMPUTING ENVIRONMENT

The Engineering Analysis and Data System (EADS) provides the Cray user at MSFC with a front-end to the supercomputer mainframe. Jobs submitted to the Cray are submitted through EADS. Figure 1 shows the system configuration for EADS.

The portion of EADS which is important to Cray/FORMA users is shown in Figure 2. Also included as part of this figure are the three general areas of concern in optimization studies for codes executing on the Cray.

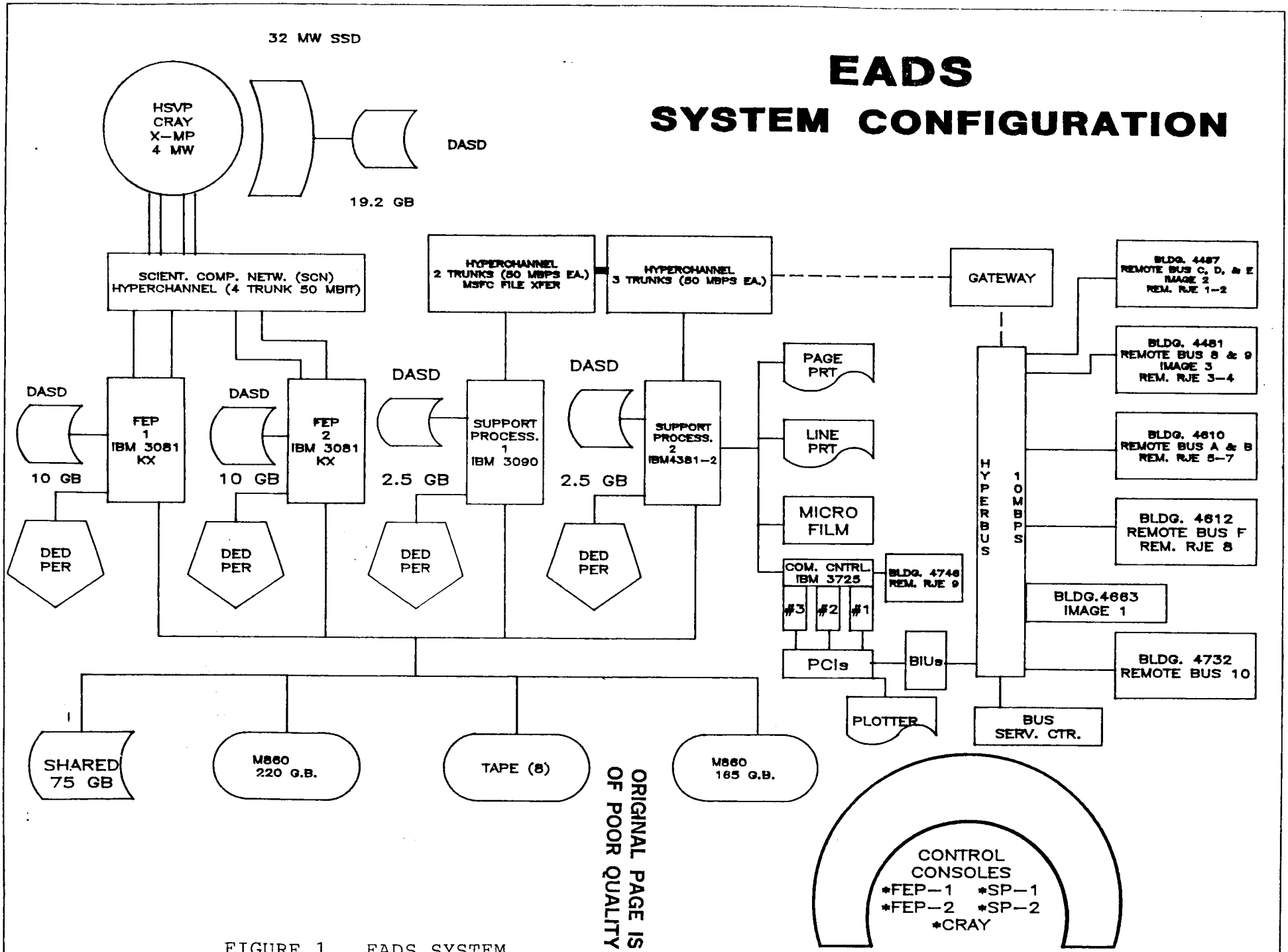
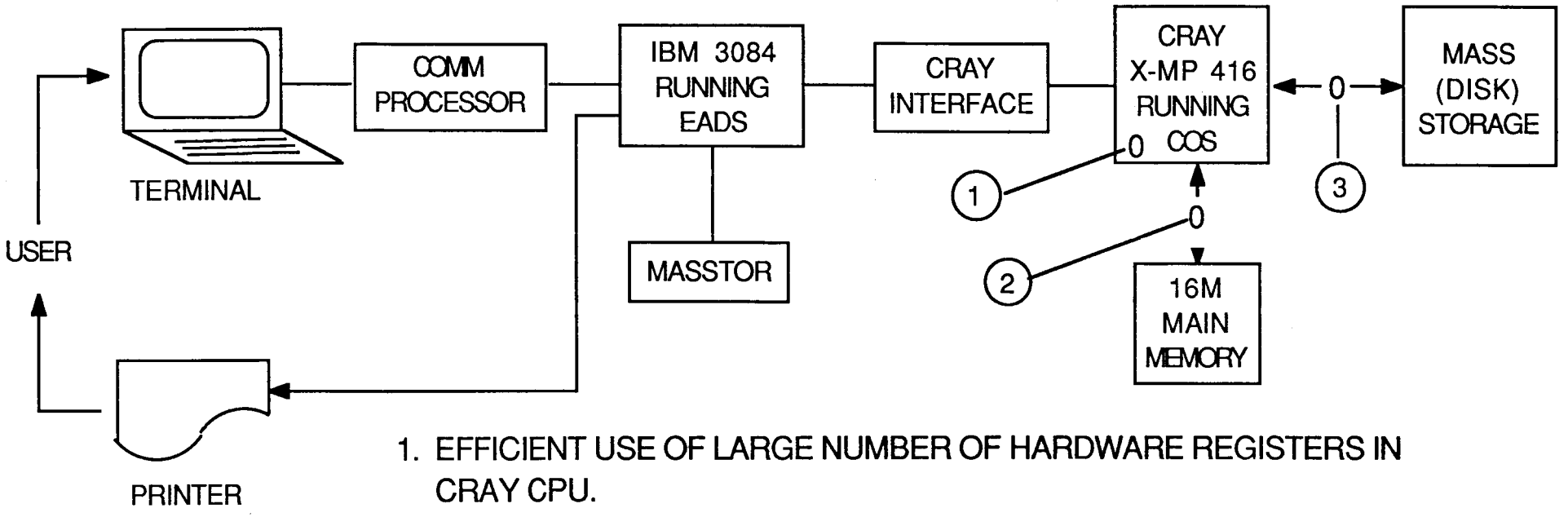


FIGURE 1. EADS SYSTEM

ORIGINAL PAGE IS
OF POOR QUALITY

THE FORMA SYSTEM ENVIRONMENT



1. EFFICIENT USE OF LARGE NUMBER OF HARDWARE REGISTERS IN CRAY CPU.

2. REDUCE THE NEED FOR MAIN MEMORY TRANSFER DURING PROGRAM EXECUTION.

3. STREAMLINE THE DATA TRANSFERS WHICH MUST TAKE PLACE BETWEEN CRAY MAIN MEMORY AND DISK STORAGE.

FIGURE 2. FORMA ENVIRONMENT

OPTIMIZATION STUDIES

The FORMA (Fortran Matrix Analysis) software package consists of the following:

105 MATRIX ANALYSIS SUBPROGRAMS:

- o 42 Arithmetic Subprograms
- o 45 Matrix Manipulation Subprograms
- o 12 I/O Utility Subprograms
- o 6 System Utility Subprograms

The FORMA subroutines are characterized by the attributes listed here:

- o MODULAR FORTRAN STRUCTURE

  The average arithmetic routine is 180 statements

  The average matrix manimulation routine is 80 statements

  The average I/O utility routine is 30 statements

  The average system utility routine is 10 statements

- o ARITHMETIC STRUCTURE

  Matrices as large as 12,000 X 12,000 are processed by using submatrices of dimension 60 X 60, plus residues

- o SUBPROGRAM DEPENDENCIES

  The average subprogram requires 5 arguments in call statement. The average subprogram call 3 other subprograms.

- VECTORIZATION

  All vectorization is presently the result of compiler-generated codes. The average subprogram contains approximately 2 vector loops set up in this fashion.

The optimization for vector processing will by very sensitive to the existing FORMA subprograms; however, the Cray X-MP architecture is equally important. Figure 3 shows the basic register configuration for the Cray X-MP. The references at the conclusion of this report provide detailed specifications on the architecture and COS operating system.

Of particular importance in the optimization process is the organization of the 8 64-word vector registers and associated vector functional units. The peak computing speeds achievable by the Cray are principally attributable to sustained vector computations.

The existing FORMA subprograms should be analyzed for the following optimization factors:

- Subroutine/function calls
- Loop indices and addressing of arrays
- Order dependencies and recursions
- use of scalars in do loops
- Decision processed
- Restructuring do loops
- General rules

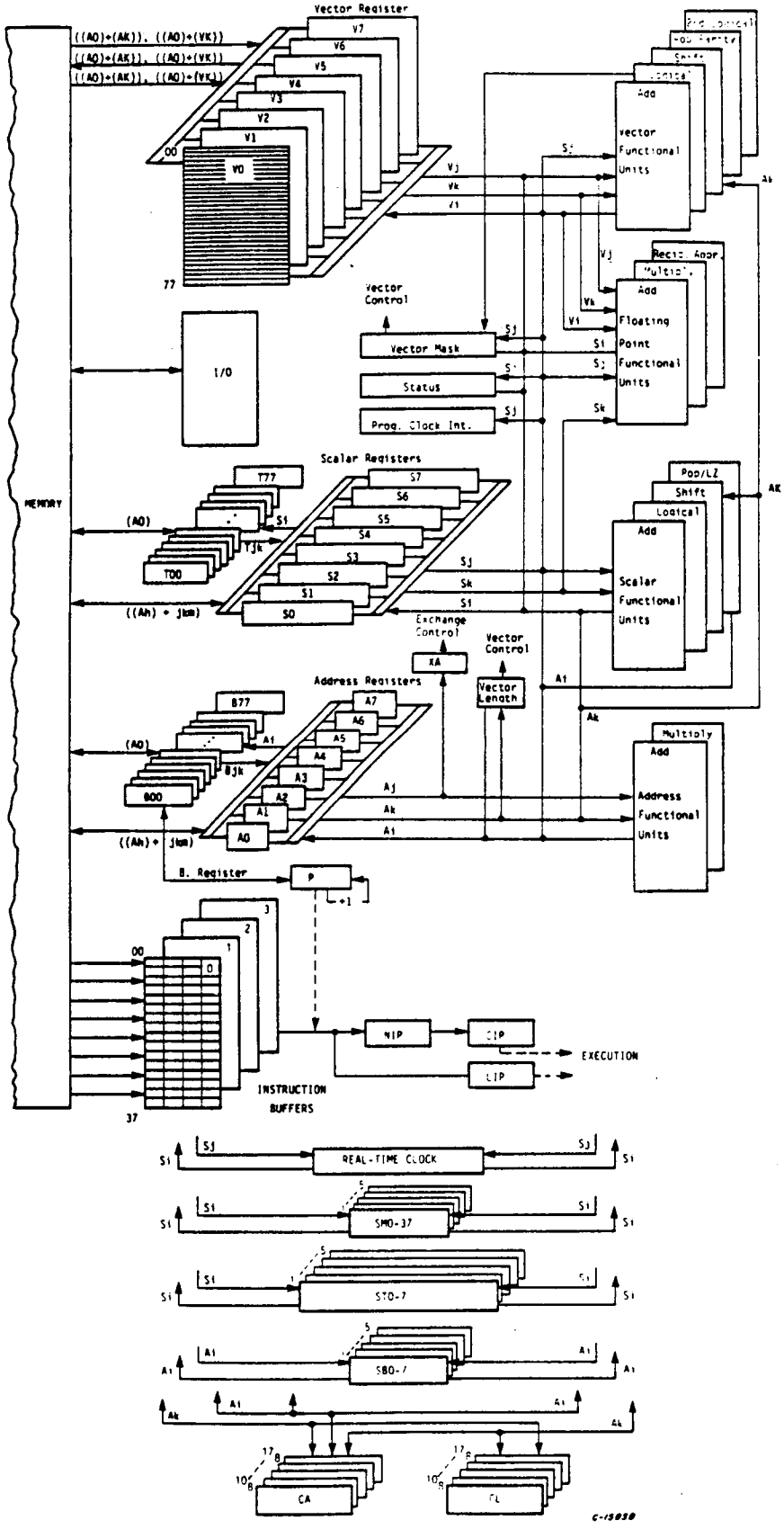


FIGURE 3. CRAY X-MP/4 BLOCK DIAGRAM

ORIGINAL PAGE IS
OF POOR QUALITY

Each of the optimization factors is now broken down into a more detailed list of do's and don'ts relative to vectorization:

CHECK GENERAL RULES

- o Avoid double precision;
- o use memory interleaving;
- o Avoid integer divides;
- o Use parentheses;
- o Avoid mixed mode expressions.

CHECK SUBROUTINE/FUNCTION CALLS

- o Isolate non-vectorizable function CALLS;
- o Separate D) loops for non-vector functions;
- o Remove (nonrecursive) SUBR CALLs from DO loops;
- o Use statement functions;
- o Convert function CALLs to user vector functions.

CHECK ORDER DEPENDENCIES-RECURSIONS:

- o Simple subscripts help compiler to recognize vectorizable loops;
- o Vectorize code on non-recursive loop indices;
- o Recognize order-dependencies--these are recursions which can be reordered to remove the dependence on order;
- o Truly recursive operations should be placed in separate DO loops;
- o Optimize when vectorize is not possible.

CHECK DECISION PROCESSES:

- o Remove loop-independent IF statements from DO loop;
- o Remove IF tests on loop indices and adjust loop bounds accordingly;
- o Create separate loops for "low-probability" decision statements involving loop indices;
- o Use temporary variable outside DO loop range for "low-probability: decision statements;
- o Avoid the computed GOTO;
- o IF-THEN-ELSE is not vectorizable;
- o Restructure conditional statements according to "density of the decision process";
- o Perform both halves of condition and then select proper results (mask undesirable ones);.

CHECK RESTRUCTURING DO LOOPS:

- o Even if additional calculations required, remove scalar statements from DO loops;
- o Use vector length of 64 whenever possible;
- o Make longer loops the innermost loops;
- o If possible, convert nested DO loops into a single DO loop;
- o Always combine DO loops of equal length;
- o "Unroll: small outer loops;
- o "Expand" small inner loops.

CHECK THE USE OF SCALARS IN DO LOOPS:

- o Check reduction functions, which result in scalars;
- o use MIN, MAX, IMIN, IMAX functions;
- o Check dot products, which result in scalars;
- o Use the SDOT functions;
- o Check matrix multiplication, whihc results in a reduction from 2 matrices to a single matrix;
- o Use matrix multiplication kernel which llows maximum vectorization (see example);
- o Convert scalar recursions to vector arrays;
- o Do not use loop indices in loop calculations.

CHECK LOOP INDICES AND ADDRESSING OF ARRAYS:

- o Check indirect addressing;
- o Avoid use of indirect addressing in generating more compact codes;
- o Use GATHER/SCATTER functions;
- o Sparse matrices are exception;
- o Whenever possible, repeated indices should have constant "stride";
- o No complicated expressions for loop indices;
- o Repeated memory references which differ by 8 or 16 locations can cause memory bank conflicts.

OPTIMIZATION STRATEGIES

There are several approaches to accomplishing the conversion of existing, non-vectorized computer codes to obtain more efficient Cray X-MP programs. In this section, a short-term strategy will be suggested and an example analysis will be discussed. In addition, a long-term conversion strategy will be outlined, along with a general optimization procedure.

Figure 4 is a flowchart of a short-term optimization procedure which addresses the conversion of more critical subprograms on a priority basis. This flowchart is specific to the FORMA software package, and when the procedure is followed for a typical job stream, we obtain the following results:

1. FORMA routines have been classified one time (this step not part of a loop) and documented, noting several key parameters and briefly describing function.

2. Typical job stream obtained from System Response Branch (ED22). This program calculates a response matrix and requires approximately 25 CPU-SEC to execute.

3. Flow trace utility provides the following statistics:

Subprogram Name	% Run-Time	Subprogram Function
RESPONS	2.33	Main Program
NTRANI	7.73	I/O Utility
NTRANR	11.00	I/O Utility
ZRDISK	3.78	I/O Utility
ZWDISK	1.80	I/O Utility
ZMULX1	35.63	[Z] = [A] * [B] + [Z]
ZMULT	28.60	[Z] = [A] * [B]
ZMAXMN	2.52	r max = max [R]
SOLVEQ	1.86	$A\frac{d^2X}{dT^2} + B\frac{dX}{dT} + CX = 0$
OTHER	4.75	36 other subprograms

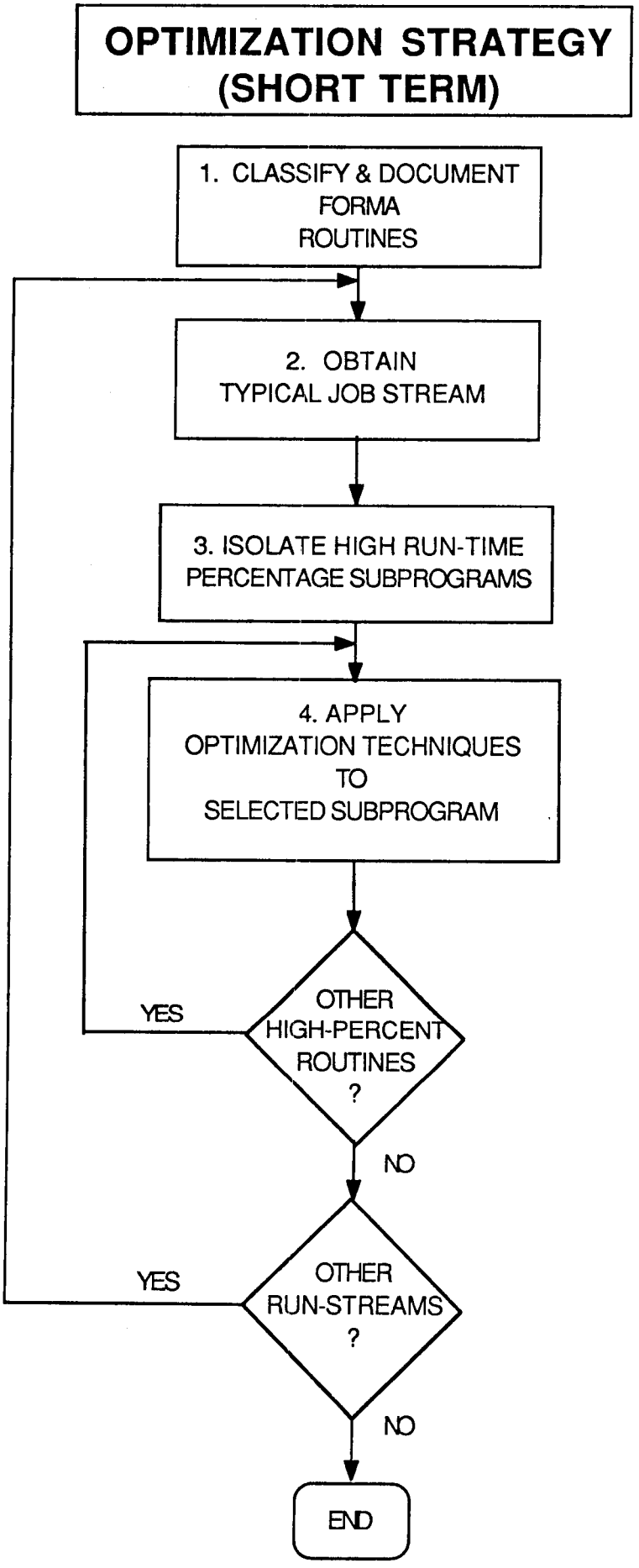


FIGURE 4. SHORT-TERM OPTIMIZATION STRATEGY

4. ZMULX1, ZMULT Optimization:

   Since these are similar routines, optimization methods will be similar;

   Restructure vector loops: one in each;

   Isolate subroutine calls, especially I/O;

   Use of scalars in DO loops;

   Vectorize decision processes, if appropriate;

   General rules.

5. We shall treat the discussion of block number 5 in the optimization strategy by showing a typical analysis process involving matrix multiplication. First, consider the "normal" matrix multiplication program segment:

```
      DO 10  I= 1,N
      DO 10  J= 1,N
      A (I,J) = 0.0
      DO 10  K= 1,N
10    A (I,J) = A(I,J) + B(I,K) * C (K,J)
```

Then consider a "better multiply kernel which allows the Cray compiler to set up more efficient vector calculations:

```
      DO 9  J=1,N
      DO 9  I=1,N
      A (I,J)=0.0
9     CONTINUE
      DO 10  K=1,N
      DO 10  J=1,N
      DO 10  I=1,N
      A(I,J)=A(I,J) + B(I,K) * C(K,J)
10    CONTINUE
```

Notice that the vectorized code is not as compact, but it allows the Cray to perform two vector calculations at the innermost loop of both nested-DO's.

Figures 5 and 6 show the ZMULT and ZMULX1 routines which were found to be the highest-run-time subprograms in our typical run stream. The reader should compare the DO loop structure discussed above with these figures.

```
1.                      SUBROUTINE ZMULX1(NMSA,NMSB,NMSZ)
2.                      DOUBLE PRECISION SA,SB,SZ
3.                      DATA KRCPRT/60/
4.                      COMMON /LZ1/ INDA(204),INDRPA(200),MHA(10),SA(60,60)
5.                      COMMON /LZ2/ INDB(204),INDRPB(200),MHB(10),SB(60,60)
6.                      COMMON /LZ3/ INDZ(204),INDRPZ(200),MHZ(10),SZ(60,60)
                    C
                    C  THIS IS A SPECIAL MULTIPLICATION SUBROUTINE DESIGNED TO BE USED
                    C  BY ZTRAE.  IT PERFORMS THE OPERATION:
                    C                    (A)*(B) + (Z) = (Z)
                    C  THIS ROUTINE ALLOWS THE NUMBER OF ROWS IN (B) TO BE LESS
                    C  THAN THE NUMBER OF COLUMNS IN (A).
                    C  CALLS FORMA SUBROUTINES:  CHKZER,DZERO,ZBEGIN,ZCLEAN,ZREDI,ZREDR,
                    C                            ZWRTI,ZWRTR,ZZBOMB.
                    C  DEVELOPED BY JOHN ADMIRE.  MAY 1981.
                    C  LAST REVISION BY JOHN ADMIRE.  JAN 1984.  (LEWIS CRAY)
                    C  IMPLEMENTED ON IBM 3084 BY DAVID S. MCGHEE. MARCH 1986.
                    C
                    C     SUBROUTINE ARGUMENTS (ALL INPUT)
                    C  NMSA   - PARTITION-LOGIC IDENT FOR MATRIX (A).
                    C  NMSB   - PARTITION-LOGIC IDENT FOR MATRIX (B).
                    C  NMSZ   - PARTITION-LOGIC IDENT FOR MATRIX (Z).
                    C
                    C     NERROR EXPLANTION
                    C  1 = NUMBER OF COLUMNS IN (A) LESS THAN NUMBER OF ROWS IN (B).
                    C  2 = NUMBER OF ROWS IN (A) NOT EQUAL TO NUMBER OF ROWS IN (Z).
                    C  3 = NUMBER OF COLUMNS IN (B) NOT EQUAL TO NUMBER OF COLUMNS IN (Z).
                    C
7.                      CALL ZBEGIN(NMSA,NRA,NCA,NRPA,NCPA,NRLA,NCLA,INDA,MHA)
8.                      CALL ZBEGIN(NMSB,NRB,NCB,NRPB,NCPB,NRLB,NCLB,INDB,MHB)
9.                      CALL ZBEGIN(NMSZ,NRZ,NCZ,NRPZ,NCPZ,NRLZ,NCLZ,INDZ,MHZ)
10.                     NERROR=1
11.                     IF(NCA .LT. NRB) GO TO 999
12.                     NERROR=2
13.                     IF(NRA .NE. NRZ) GO TO 999
14.                     NERROR=3
15.                     IF(NCB .NE. NCZ) GO TO 999
16.     :---------------        DO 150 IRPA=1,NRPA
17.     :                       NRSA=KRCPRT
18.     :                       IF(IRPA .EQ. NRPA) NRSA=NRLA
19.     :                       CALL ZREDI(INDRPA,200,INDA(IRPA))
20.     :                       CALL ZREDI(INDRPZ,200,INDZ(IRPA))
21.     :  :------------       DO 140 JCPB=1,NCPB
22.     :  :                    NCSB=KRCPRT
23.     :  :                    IF(JCPB .EQ. NCPB) NCSB=NCLB
24.     :  :                    IF(INDRPZ(JCPB) .LE. 0) GO TO 100
25.     :  :                    CALL ZREDR(SZ,KRCPRT*KRCPRT,INDRPZ(JCPB))
26.     :  :                    GO TO 110
27.     :  :                100 CALL DZERO(SZ,NRSA,NCSB,KRCPRT)
28.     :  :                110 CONTINUE
29.     :  :  :---------       DO 130 IRPB=1,NRPB
30.     :  :  :                 NCSA=KRCPRT
31.     :  :  :                 IF(IRPB .EQ. NRPB) NCSA=NRLB
32.     :  :  :                 IF(INDRPA(IRPB) .LE. 0) GO TO 130
33.     :  :  :                 CALL ZREDI(INDRPB,200,INDB(IRPB))
34.     :  :  :                 IF(INDRPB(JCPB) .LE. 0) GO TO 130
35.     :  :  :                 CALL ZREDR(SA,KRCPRT*KRCPRT,INDRPA(IRPB))
36.     :  :  :                 CALL ZREDR(SB,KRCPRT*KRCPRT,INDRPB(JCPB))
37.     :  :  :  :------        DO 120 I=1,NRSA
38.     :  :  :  :  :---        DO 120 J=1,NCSB
39.     :  :  :  :  :  Rp-      DO 120 L=1,NCSA
40.     :  :  :  :--:--Rp-  120 SZ(I,J)=SZ(I,J)+SA(I,L)*SB(L,J)
LX1   NOVECTOR - REPLACED BY CALL TO '$SDOT' ********************************P=0005045C
41.     :  :  :---------       130 CONTINUE
42.     :  :                    CALL CHKZER(SZ,NRSA,NCSB,IFZERO,KRCPRT)
43.     :  :                    IF(IFZERO .LE. 0 .AND. INDRPZ(JCPB) .GT. 0)
        :  :                   *INDRPZ(JCPB)=-INDRPZ(JCPB)
44.     :  :                    IF(IFZERO .GT. 0)
        :  :                   *CALL ZWRTR(SZ,KRCPRT*KRCPRT,INDRPZ(JCPB))
45.     :  :------------       140 CONTINUE
46.     :---------------       150 CALL ZWRTI(INDRPZ,200,INDZ(IRPA))
47.                             CALL ZCLEAN(NMSZ,INDZ,MHZ)
48.                             RETURN
49.                         999 CALL ZZBOMB(' ZMULX1 ',NERROR)
50.                             END
    ONE LINE DO LOOP REPLACED AT SEQ. NO.        39; P=      201B
```

FIGURE 5. ZMULX1 SOURCE CODE

```
1.                      SUBROUTINE ZMULT  (NMSA,NMSB,NMSZ)
2.                      DOUBLE PRECISION SA,SB,SZ,S,SS
3.                      COMMON /LZ1/ INDA(204),INDRPA(200),MHA(10),SA(60,60)
4.                      COMMON /LZ2/ INDB(204),INDRPB(200),MHB(10),SB(60,60)
5.                      COMMON /LZ3/ INDZ(204),INDRPZ(200),MHZ(10),SZ(60,60)
6.                      DIMENSION X(60,60)
7.                      DATA KRCPRT/60/
                    C
                    C   MATRIX MULTIPLICATION FOR PARTITION-LOGIC. (A) * (B) = (Z).
                    C   CALLS FORMA SUBROUTINES CHKZER,ZBEGIN,ZCLEAN,ZOPEN,ZREDI,
                    C                          ZREDR,ZWRTI,ZWRTR,ZZBOMB.
                    C   DEVELOPED BY RL WOHLEN. AUGUST 1977.
                    C   LAST REVISION BY JOHN ADMIRE.  FEB 1982.
                    C   IMPLEMENTED ON IBM 3084 BY DAVID S. MCGHEE. MARCH 1986.
                    C
                    C       SUBROUTINE ARGUMENTS (ALL INPUT)
                    C   NMSA    = PARTITION-LOGIC NAME FOR MATRIX (A).
                    C   NMSB    = PARTITION-LOGIC NAME FOR MATRIX (B).
                    C   NMSZ    = PARTITION-LOGIC NAME FOR MATRIX (Z).
                    C
                    C       NERROR EXPLANATION
                    C   1 = MATRICES (A) AND (B) ARE NOT COMPATIBLE SIZE.
                    C
                    C   READ MATRIX (A) HEADER.
8.                      CALL ZBEGIN(NMSA,NRA,NCA,NRPA,NCPA,NRLA,NCLA,INDA,MHA)
                    C
                    C   READ MATRIX (B) HEADER.
9.                      CALL ZBEGIN(NMSB,NRB,NCB,NRPB,NCPB,NRLB,NCLB,INDB,MHB)
                    C
                    C   CHECK (A) AND (B) MATRICES FOR SIZE COMPATIBILITY.
10.                                                                    NERROR=
11.                     IF (NCA .NE. NRB) GO TO 999
                    C
                    C   FORM MATRIX (Z) HEADER.
12.                     NRZ=NRA
13.                     NCZ=NCB
14.                     CALL ZOPEN(NMSZ,NRZ,NCZ,NRPZ,NCPZ,NRLZ,NCLZ,INDZ,MHZ)
                    C
                    C   MULTIPLY MATRICES (A) AND (B).
15.     :-----------------      DO 29 IRPA=1,NRPA
16.     :                       NRSA=KRCPRT
17.     :                       IF(IRPA .EQ. NRPA) NRSA=NRLA
18.     :                       CALL ZREDI(INDRPA,200,INDA(IRPA))
19.     :                       CALL ZREDI(INDRPZ,200,INDZ(IRPA))
20.     :  :--------------      DO 25 JCPB=1,NCPB
21.     :  :                    NCSB=KRCPRT
22.     :  :                    IF(JCPB .EQ. NCPB) NCSB=NCLB
23.     :  :                    CALL ZREDR(SZ,KRCPRT*KRCPRT,INDRPZ(JCPB))
24.     :  :                    IFSZ=0
25.     :  :  :-----------      DO 24 JCPA=1,NCPA
26.     :  :  :                 NCSA=KRCPRT
27.     :  :  :                 IF(JCPA .EQ. NCPA) NCSA=NCLA
28.     :  :  :                 IF(INDRPA(JCPA) .LE. 0) GO TO 24
29.     :  :  :                 CALL ZREDI(INDRPB,200,INDB(JCPA))
30.     :  :  :                 IF(INDRPB(JCPB) .LE. 0) GO TO 24
31.     :  :  :                 IFSZ=1
32.     :  :  :                 CALL ZREDR(SA,KRCPRT*KRCPRT,INDRPA(JCPA))
33.     :  :  :                 CALL ZREDR(SB,KRCPRT*KRCPRT,INDRPB(JCPB))
        :  :  :             C
34.     :  :  :                 IF(NRSA.NE.KRCPRT .OR. NCSA.NE.KRCPRT .OR. NCSB.NE.KRCPRT)THEN
35.     :  :  :  :--------      DO 5000 J=1,NCSB
36.     :  :  :  :  :-----        DO 5000 K=1,NCSA
37.     :  :  :  :  :  V--           DO 5000 I=1,NRSA
38.     :  :  :  :  :  V                SZ(I,J)=SZ(I,J)+SA(I,K)*SB(K,J)
39.     :  :  :  :--:--V--  5000  CONTINUE
40.     :  :  :                 ELSE
41.     :  :  :                   CALL MXM(SA,KRCPRT,SB,KRCPRT,X,KRCPRT)
42.     :  :  :     :-----        DO 5010 J=1,NCSB
43.     :  :  :     :  V--           DO 5010 I=1,NRSA
44.     :  :  :     :  V                SZ(I,J)=SZ(I,J)+X(I,J)
45.     :  :  :     :--V--  5010    CONTINUE
46.     :  :  :                 END IF
        :  :  :             C
47.     :  :  :-------------   24 CONTINUE
48.     :  :                    IF(IFSZ .EQ. 0) GO TO 26
49.     :  :                    CALL CHKZER(SZ,NRSA,NCSB,IFZERO,KRCPRT)
50.     :  :                    IF(IFZERO .EQ. 0) GO TO 26
51.     :  :                    CALL ZWRTR(SZ,KRCPRT*KRCPRT,INDRPZ(JCPB))
52.     :  :--------------      26 CONTINUE
        :                   C
53.     :-----------------      29 CALL ZWRTI(INDRPZ,200,INDZ(IRPA))
54.                             CALL ZCLEAN(NMSZ,INDZ,MHZ)
                    C
55.                             RETURN
                    C
```

FIGURE 6. ZMULT SOURCE CODE

ORIGINAL PAGE IS OF POOR QUALITY

The reader should also note that the Cray compiler has provided printout information showing all program loops, which are very important in the vectorization process. The compiler also marks each loop to inform the user of the vectorization which can be obtained, i.e., fully vectorized, conditionally vectorized, short vector loop, or a vector loop replaced by a subroutine call.

In examining Figures 5 and 6, it should be noted that, even for highly modular programs, the application of all vectorization rules which have been pointed out is a very tedious process. The vectorizing compiler provided by Cray, CFT or CFT77, performs well in finding vector constructs; however, it cannot perform as well as the vector programmer who carefully examines and optimizes codes to fully exploit the X-MP architecture. The following estimates conclude this example by calculating overall run-time improvement for RESPONS if the stated levels of improvement are achieved for subprograms:

Estimate 25% improvement in ZMULX1

Estimate 25% improvement in ZMULT

Estimate 15% improvement in the other six predominant subroutines

This yields and estimated overall improvement of

$$(0.25)\ (0.64) + (0.15)\ (0.29) \simeq 0.20,$$

or 20% improvement in a typical run stream.

Figure 7 shows a long-term strategy which could be employed if a complete conversion to vectorized code is justifiable for the FORMA package. This flow chart represents a procedure which would be a greater expense and requires more time, but which would yield a thorough redesign of the software.

A general optimization strategy is shown by the flow chart of Figure 8. This procedure is independent of the specific software package under consideration. Note that the procedure would require the implmentation of general purpose test and data generation programs to thoroughly test vectorization strategies.

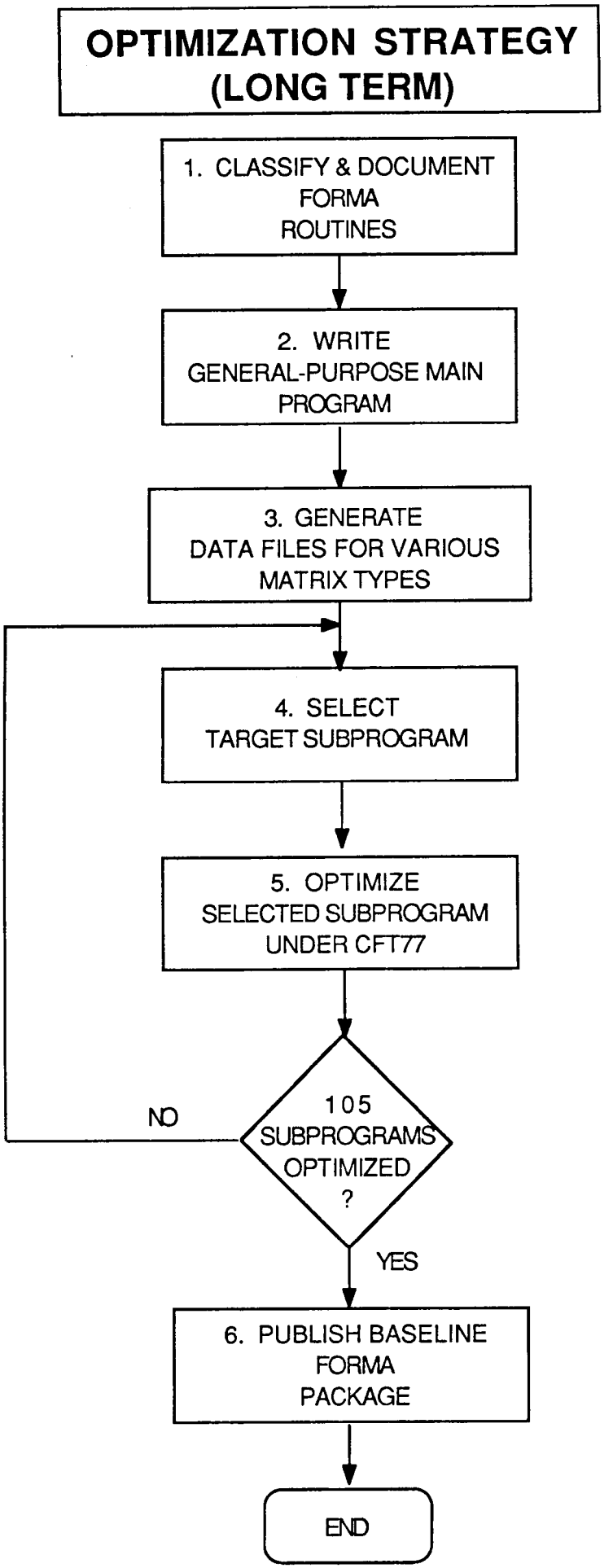


FIGURE 7. LONG-TERM STRATEGY

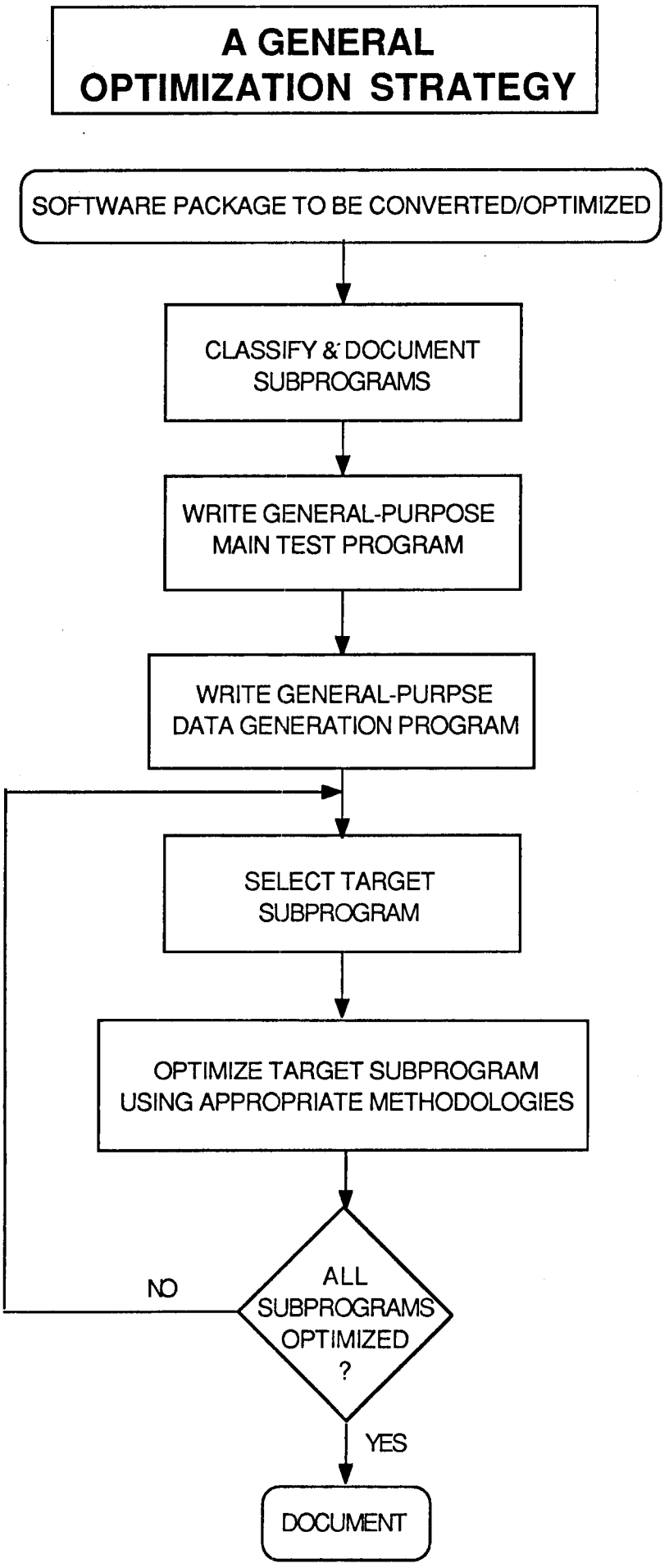


FIGURE 8. GENERAL STRATEGY

CONCLUSIONS AND FUTURE DIRECTION

Optimization of computer programs to achieve highly vectorized codes is a very exacting and time-consuming process. It is very much labor-intensive and it requires highly skilled personnel. On the other hand, these are rather costly attributes that must be balanced against the fact that software such as the FORMA routines are long-term investments. There are high initial costs associated with the optimization process, but there are long-term advantages to reducing CPU-minutes for frequently used programs.

The FORMA software package would be an excellent candidate for long-term optimization procedures. If this is done, several key areas would need to be addressed. These are:

- The CFT77 compiler should be used in generating object code. In doing this, complied codes should be compared with previous compilations to ensure the integrity of the compile process.

- I/O utility routines are not particularly good candidates for optimization. However, these are frequently used routines and unique I/O speed-up features on the Cray should be investigated. These would include BUFFER IN/BUFFER OUT and unformatted I/O.

- Custom performance monitoring routines should be implemented. These could provide users with a means to easily monitor performance enhancements and to monitor any difference in results obtained.

- The optimization techniques which are effective tend to be reusable; that is, once learned or recognized, the same techniques can generally be applied a number of times in a given software package. Therefore, the more effective vectorization techniques should be well documented, including applicable performance statistics.

The Cray X-MP at NASA/MSFC represents a significant investment in high-performance computing technology. As such, resources to support this machine are critical. Those personnel writing new programs for the Cray X-MP should be well-versed in good vectorization techniques. In addition, permanent staff with in-depth knowledge of vectorization tools and techniques is important to the effective use of the present machine, as well as future upgrades and next-generation machines.

REFERENCES

FORTRAN Programming on Cray Computers, Student Workbook, Pacific-Sierra Corp.

Cray X-MP and Cray-1 Computer Systems, Programmer's Library Reference Manual, Cray Research, Inc., 1987.

Cray X-MP and Cray-1 Computer Systems, FORTRAN (CFT) Reference Manual, Cray Research, Inc., 1986.

EADS User's Guide, Grumman Data Systems, Doc. No. GDS-MSFC-001, 1987.

Cray X-MP and Cray-1 Computer Systems COS Version 1 Reference Manual and COS Version 1 Ready Reference Manual, Cray Research, Inc., 1987.

Computer Architecture and Parallel Processing, Hwang and Briggs, McGraw-Hill, 1986.

"Modal Analysis of Structures by an Iterative Rayleigh - Ritz Technique", NASA TM X-64528, 1970.